Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Discrete Applied Mathematics 129 (2003) 83–98
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Inverse optimization in high-speed networks

András Faragó[a,*], Áron Szentesi[b], Balázs Szviatovszki[b]

[a] *Erik Jonsson School of Engineering and Computer Science, The University of Texas at Dallas, MS-EC31, P.O. Box 830688, Richardson, TX 75083-0688, USA*
[b] *Ericsson Traffic Lab, Budapest, P.O. Box 107, Budapest, H-1300, Hungary*

Abstract

A general approach is presented for handling the following *inverse optimization* problem: given solutions to each member of a family of combinatorial optimization tasks on a common underlying set, find a positive linear objective function (weighting) on the common underlying set that simultaneously makes each solution optimal in its own optimization task. Our motivation stems from the inverse shortest path problem that is made practically important in high-speed telecommunication networks by the Asynchronous Transfer Mode Forum's Private Network–Network Interface architecture, in which route finding can be based on administrative weights. Different variants of the problem are investigated, including uniqueness requirements and reserve routes.

1. Introduction

There is recently an increasing interest in *inverse optimization* problems related to combinatorial optimization and linear programming. The most general formulation of the tasks that have been studied so far is given by Ahuja and Orlin [2], where several motivating applications are also listed (our motivating application is rooted in telecommunications and will be detailed in Section 2). The following general inverse optimization problem is considered by Ahuja and Orlin [2]. Given a set S of feasible vectors (solutions), a cost vector c and a vector $x_0 \in S$, find a new cost vector d such that x_0 becomes an optimal solution with respect to the new cost and the perturbation $\|d-c\|$ of the original cost is minimum in terms of some norm. This is the "inverse" of the original optimization problem $\max\{cx \mid x \in S\}$, in the sense that now the solution

* Corresponding author.
E-mail addresses: farago@utdallas.edu (A. Faragó), ethasi@eth.ericsson.se (A. Szentesi), ethbsii@eth.ericsson.se (B. Szviatovszki).

0166-218X/03/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0166-218X(02)00235-4

x_0 is given, which may not be optimal with respect to the original cost c, and the goal is to find a minimal perturbation of the cost vector that makes x_0 an optimal solution.

The most general result in the above direction is that if the original optimization problem is solvable in polynomial time, then this inverse is also polynomially solvable when the deviation of the cost vectors is measured by the L_1 or L_∞ norm [2]. The general solution is based on the Ellipsoid Algorithm with a separation oracle. On the other hand, in most of the interesting cases the original optimization problem has some special structure which can be further exploited to obtain more efficient algorithms for specific problems. Utilizing the structure, a number of papers addressed special inverse optimization problems, for example (based on the overview in [2]): inverse linear programming [2], inverse shortest paths [4–7,23,26], inverse spanning tree [16,21,3], inverse sorting [1], inverse shortest arborescence [12], inverse bipartite k-matching [13], inverse minimum cut [24,25], inverse minimum cost flow [14,20], inverse matroid intersection [15], inverse polymatroidal flow [8].

Note that in the above problems, especially in the purely combinatorial settings, the difficulty typically stems from the fact that a *minimal perturbation* of the original cost vector is searched for. Otherwise, the problem would trivialize in most cases. For example, finding just *any* cost vector that makes, say, a given spanning tree optimal could easily be achieved by assigning small cost to the tree edges and large costs to all others.

In the family of inverse problems that we consider in this paper the source of difficulty is different. We do not have an original cost function from which the deviation is to be minimized. On the other hand, we have a *family* of combinatorial optimization problems with different sets of feasible solutions over the same underlying finite set and a given solution is specified for each. We are looking for a positive weighting of the common underlying set that makes *each* of the specified solutions *simultaneously* optimal for its own problem. Thus, the difficulty is that while for one optimization task one could easily make a given solution optimal, those weights may be poor for some others. We also consider two further variants: (1) the issue of making the specified solutions *unique* optimums, and, (2) how to find a cost vector that works approximately, in the sense of minimizing the maximum absolute deviation from the optimum in the respective tasks, if no cost vector exists that makes each prescribed solution optimal.

The general framework is motivated by inverse shortest path problems that are raised in the context of Asynchronous Transfer Mode (ATM) networks (see Section 2). Below we define three such tasks.

As input we are given an undirected graph $G = (V, E)$ with vertex and edge sets V, E, respectively, with no parallel edges and loops. Also given a set $\mathscr{P} = \{P_{ij}\}$ of paths in the graph between certain (but not necessarily all) pairs (i, j) of nodes. We want to assign positive weights to the edges of the graph. The weights are to be chosen such that with respect to these weights the given paths arise as minimum weight paths between their endpoints. The weights are restricted to be strictly positive to exclude the trivial case when each weight is 0, since then every path would be a minimum weight path with zero total weight. Below we list three different versions of

this inverse shortest path problem. Each specifies a different task, in increasing order of difficulty:

Task I: Find the weights such that each given route in $\mathscr{P}$ arises as a minimum weight path between its endpoints.

Task II: Find the weights such that each given route in $\mathscr{P}$ arises as a *unique* minimum weight path between its endpoints. (That is, for each given path $P_{ij} \in \mathscr{P}$, any other path between i and j has strictly larger weight.)

Task III: Assume that $\mathscr{P}$ is such that for each P_{ij} it also contains another path Q_{ij} between the same endpoints. P_{ij} and Q_{ij} are assumed edge-disjoint and their intended meaning is that they are the primary and reserve routes, respectively, between nodes i, j. Find the weights such that each given route P_{ij} arises as a minimum weight path between its endpoints. Moreover, if any edge of P_{ij} is removed from the graph, then, under the same weights, Q_{ij} becomes a minimum weight path between i and j, while any path $P_{kl} \in \mathscr{P}$ that does not contain the removed edge remains the shortest path. This is required to hold for any choice of i, j, k, l with the same weights.

In Section 3 we present a general solution which contains all these and many other possible tasks as special cases, after reviewing the telecommunications-based motivation in Section 2. In Section 4 we consider the case when no weighting exists that can make each given solution exactly optimal, so we search for a weighting which minimizes the maximum difference between the actual weights of the prescribed solutions and the optimal ones. In Section 5 we present a case study concerning the unique inverse shortest path problem (Task II). As a first step of the case study, we show that using the special structure it can be solved by a simpler and more direct LP formulation than the one which would follow from the general framework. Then we apply the solution to demonstrate that one can achieve improved network performance in a realistic scenario. This is done via a simulated hypothetical pan-European ATM network with randomly generated traffic.

2. Networking motivation

ATM is generally accepted as an efficient and standardized transfer mode that provides the transport network technology for emerging integrated networks, carrying a wide scale of services, including voice, data and video. The flexibility and efficient bandwidth utilization capability of ATM makes it suitable to support service integration both in wired and wireless networks. For these networks a general framework to provide a scalable, hierarchically organized routing and signalling solution was introduced by the ATM Forum. This is the *Private Network–Network Interface* (*PNNI*) *Specification* [17] that has the main objectives of supporting *Quality of Service* (*QoS*) based, scalable routing with as much *auto-configuration* as possible.

In PNNI networks no routing information has to be configured manually when adding a new node or link to an existing network. With the help of a dynamic, link-state flooding protocol, significant changes in link parameters (availability, load, delay or delay variation) are distributed automatically to other nodes in the network. Based on the flooded information each node builds a link-state database, which represents the

network as a graph with several parameters on each link. The path-selection algorithm operates on this topology database to compute the routes for traffic sources. In PNNI if a call establishment fails, it is handed back to the originating node that can make more trials. To benefit from this crankback procedure the path-selection algorithm is expected to provide more than one path between a source and a destination.

By adapting automatically to the actual traffic situation, the PNNI dynamic routing framework can entirely take over the task of managing routes for traffic through the network both at the design and management phase of the system. This is an essential advantage of PNNI.

On the other hand, the practical implementation of such a dynamic routing mechanism is far from trivial. There are a number of concerns that arise in the practical implementations. For example, since dynamic routing inherently contains several control and feedback loops, stability is an important concern. Moreover, the use of existing network design and dimensioning methods become questionable when dynamic routing is brought into the picture.

These potential drawbacks may deter operators from the deployment of a truly dynamic routing in ATM networks, since the operators naturally prefer solutions that have been thoroughly tested, that are proven to be stable, and for which there exist well-tried, reliable control, design and dimensioning methods. For this reason, the ATM Forum provides an interim solution that combines the simplicity of fixed routing with some of the advantages of a dynamic scheme. This is achieved by the use of manually configured *Administrative Weights* that are assigned to the links to express the potentially complex link state information in an aggregated way.

With the use of Administrative Weights the route selection is simplified to finding shortest paths by well-known algorithms, such as Dijkstra's. Via configuring the weights the operator can direct the traffic in the network similar to the fixed routing paradigm, but eliminating the maintenance of large routing tables and benefiting from a reliable, self-configuring system without stability problems. This solution provides a smooth transition towards fully dynamic routing, and, indeed, all vendors of PNNI equipment support it. In this way, one can design the flow of traffic in the dynamic PNNI environment similar to the static fixed routing and utilize the existing network design methodology for the new technology.

A key question in implementing this strategy is: how to set the Administrative Weights in a network? This naturally leads to the inverse shortest path problem mentioned in the Introduction. Thus, one can design the traffic flow as in a network with fixed routing (see e.g. [9,18,10]) and then, via solving the inverse shortest path problem, the administrative weights can be obtained.

Beyond the above simplest version, there are other considerations that require more complex versions of the inverse shortest path problem in large networks. One of them is the issue of uniqueness. If the Administrative Weights are set such that they allow several different minimum weight paths between the same end nodes, then the equipment may not choose the one which was chosen as desirable in the design phase. This can severely deteriorate the network throughput, since the network is dimensioned on the assumption that the designed paths are used. This leads to the unique inverse shortest path problem, where the weights are to be chosen such that the prescribed paths

become *unique* shortest paths between their endpoints. At this point it is worth mentioning that if we only want to design *any* route system that provides unique shortest paths under some weighting, then it is easily achieved by finding shortest paths using weights given by $w_i = 2^{m+1} + 2^i$ for edge i, where m is the number of edges. Such a solution however, may not satisfy other route design criteria. For example, it has poor load-balancing properties, as edges with small index will always be preferred, possibly leading to disproportionate overload on these links, while leaving others with high index underutilized.

Another more complex concern is network reliability and survivability. In a large network that carries high-traffic volume, the failure of a link or node may result in an essential loss of revenue. To avoid this, for every traffic route a reserve route is planned and whenever the primary route is not available, due to e.g. a link failure, the system will automatically select the secondary (reserve) route. If we want to incorporate this into the PNNI framework with Administrative Weights, then we obtain a new inverse shortest path task version. This is given by specifying two disjoint paths for each traffic origin–destination pair, one is the primary, the other is the reserve route. We want to set the weights such that all primary routes become shortest paths when all links are in operational state. On the other hand, if any given link disappears from the network due to failure, then we require that the same weights will automatically turn the reserve routes into shortest paths between those node pairs for which the primary routes are not available because they contain the failed link, while the available primary routes still remain the shortest paths.

3. A general framework

In this section we introduce a general model that contains all the inverse shortest path problems, as well as many other inverse optimization tasks as special cases.

Let H be a finite set. Assume there are certain set systems $\mathscr{F}_1, \ldots, \mathscr{F}_m$ over the underlying set H, that is, each $\mathscr{F}_i$ is a family of subsets of H. Assume further that for each $\mathscr{F}_i$ an algorithm $\mathscr{A}_i$ is given. The algorithm $\mathscr{A}_i$ can do the following. As input it receives a weight vector w that defines a weighting of the underlying set H. As a result, the algorithm returns a set $A \in \mathscr{F}_i$ that has minimum weight among the sets in $\mathscr{F}_i$, that is, $w(A) \leqslant w(A')$ holds for any $A' \in \mathscr{F}_i$, where $w(A)$ denotes the weight of A. The relationship that A is such a minimum weight set is shortly denoted by $w(A) = \mathscr{A}_i(w)$, where $\mathscr{A}_i(w)$ is the weight of the set returned by the algorithm.

A first general formulation of the (non-unique) inverse optimization problem, using the above introduced notations, can be given as follows (later this will be refined to incorporate further cases):

Task A

Input: Subsets $A_1 \in \mathscr{F}_1, \ldots, A_m \in \mathscr{F}_m$.

Find: A positive weight vector w, such that $w(A_i) = \mathscr{A}_i(w)$ holds for every $i = 1, \ldots, m$.

Let us see first how the non-unique inverse shortest path problems fit in this framework. For Task I of the Introduction we take H as the set of all edges in the graph. For each origin-destination pair $p = (i, j)$ we define a set system $\mathscr{F}_p$ such that a set F is

in $\mathscr{F}_p$ if and only if F is the set of edges in a simple (=loop-free) path between i and j. The associated algorithm $\mathscr{A}_p$ is any shortest path algorithm that finds a minimum weight path between i and j. Finally, the sets $A_1,\ldots,A_m \subseteq H$ are the sets of edges in the prescribed paths P_{ij} that are to be made shortest paths. In this simple case, it is clear that these assignments specialize Task A exactly into Task I.

It is important to note that the set systems need not be given explicitly, as they usually contain exponentially many sets. It will be clear from the solution later, that each set system is represented by the corresponding algorithm only and no other representation is needed.

For Task III we start similarly, but here, in addition to the set systems used for Task I, we introduce more set systems that are indexed by triples. Let $q=(i,j,e)$ be a triple in which i,j denote nodes and e runs over the edges. If $p=(i,j)$, then let $\mathscr{F}_p$ and $\mathscr{A}_p$ be the same as for Task I above. For $q=(i,j,e)$ let $F\in\mathscr{F}_q$ if and only if F is the set of edges in a simple $i-j$ path that does not contain the edge e. The corresponding algorithm $\mathscr{A}_q$ is any shortest path algorithm that finds a minimum weight path between i and j, such that the path does not contain e (i.e., it finds the shortest path in the graph with e deleted). Finally, the sets $A_1,\ldots,A_m \subseteq H$ are sets of edges in the prescribed paths, such that for each primary path an edge-disjoint reserve path is also given. Again, a little thought can easily convince the reader that with this setting Task A specializes exactly to Task III.

Using this framework we can define many other (non-unique) inverse shortest path problems. For example, one can take for $\mathscr{F}_p$ with $p=(i,j)$ the sets F that are sets of edges in simple $i-j$ paths with the restriction that they do not contain more edges than a given constant. A networking motivation for this is a restriction on the number of hops to satisfy e.g. delay constraints. The corresponding algorithm finds minimum weight paths among these restricted paths. Even if this is algorithmically difficult (being NP-hard), the general framework and the inverse problem, as defined generally above, still makes sense.

With other restrictions many other versions arise and quite a few makes sense from the telecommunications point of view, as well. For example, we can restrict ourselves to the paths that contain the minimum number of edges from a given subset. This is motivated by the fact that certain network links may be overloaded. Naturally, those routes are preferred that contain the minimum number of overloaded links and we search for a minimum weight path only among these. Another restriction is to consider only those paths that do not exceed certain prescribed weights under other given weightings. This allows to represent requirements on quality of service parameters. All these and a good number of other cases can easily be cast in our general framework, just as Tasks I and III above.

Thus, Task A is a common generalization of many inverse problems that do not demand unique solutions. Let us now introduce further refinement by requiring unique solutions for some, but not necessarily all, set systems. For easy definition let us use the notation $A=\tilde{\mathscr{A}}(w)$ to denote that A is a minimum weight set under the weighting w and it is unique, i.e., there is no other set in the corresponding set system with the same weight (that is, the algorithm can only return A). Now we can define the problem that accounts for uniqueness, as well.

Task B

Input: Subsets $A_1 \in \mathscr{F}_1, \ldots, A_m \in \mathscr{F}_m$ and an integer i_0 with $0 \leqslant i_0 \leqslant m$.

Find: A positive weight vector w, such that $w(A_i) = \mathscr{A}_i(w)$ holds for every $i = 1, \ldots, m$ and for the first i_0 indices $A_i = \tilde{\mathscr{A}}_i(w)$ also holds.

In search for a solution it is enough to restrict ourselves to Task B, as it contains Task A as a special case, since with $i_0 = 0$ we directly get back to Task A. Task B, however, allows to require uniqueness for some (or possibly all) subsets. Thus, beyond containing Tasks I and III, it already incorporates Task II, as well. The solution is given by the following theorem.

Theorem 1. *There exists an algorithm $\mathscr{A}$ with the following properties*:

- *Using the algorithms $\mathscr{A}_1, \ldots, \mathscr{A}_m$ as subroutines, algorithm $\mathscr{A}$ finds a weighting w that satisfies the conditions of Task B if such a weighting exists, otherwise declares that there is no solution.*
- *If any call to $\mathscr{A}_i$, $i = 1, \ldots, m$, is counted as a single step, then algorithm $\mathscr{A}$ runs in polynomial time.*

Before proving the theorem it is worth mentioning that the complexity of the algorithms $\mathscr{A}_1, \ldots, \mathscr{A}_m$ is irrelevant here, since they are considered "black boxes": subroutines which run in unit time whenever called, independently of what computations they do inside. In the language of theoretical computer science $\mathscr{A}_1, \ldots, \mathscr{A}_m$ are used as *oracles* and the solving algorithm $\mathscr{A}$ runs in *oracle polynomial time*, using these oracles. As an immediate consequence we obtain the following important corollary.

Corollary 1. *If the algorithms $\mathscr{A}_1, \ldots, \mathscr{A}_m$ run in polynomial time, then a weighting w that satisfies the conditions of Task B can be found in polynomial time, if such a w exists.*

This means, if the original problems can be solved in polynomial time, then the inverse can also be solved, both with and without uniqueness requirements. Now we can turn to the proof of the theorem.

Proof of Theorem 1. Let us use the notations $H = \{1, \ldots, n\}$ and $w = (w_1, \ldots, w_n)$. Furthermore, let $s(x)$ be a function defined by $s(x) = 1$ if $x \geqslant 0$ and $s(x) = 0$ if $x < 0$. Now let us consider the following system of linear inequalities:

$$w_i \geqslant 1 \quad (i = 1, \ldots, n), \tag{1}$$

$$\sum_{j \in F} w_j - \sum_{j \in A_k} w_j \geqslant s(i_0 - k) \quad (\forall F \in \mathscr{F}_k - \{A_k\}, k = 1, \ldots, m). \tag{2}$$

We show first that this system of linear inequalities has a solution if and only if there exists a weighting w that satisfies the conditions of Task B. To show this, assume that there is a vector w that satisfies (1) and (2). Then by (1) w is a positive weighting. For any $k > i_0$ (which can occur only if $i_0 < m$) from (2) we have

$$\sum_{j \in F} w_j - \sum_{j \in A_k} w_j = w(F) - w(A_k) \geqslant 0$$

since $s(i_0 - k) = 0$ holds for $k > i_0$. Hence, for $k > i_0$ we have $w(A_k) \leqslant w(F)$ for every $F \in \mathscr{F}_k$, that is, A_k is a minimum weight set in $\mathscr{F}_k$. For the case $k \leqslant i_0$ (which can occur only if $i_0 > 0$), using that now $s(i_0 - k) = 1$ holds, we similarly obtain $w(A_k) \leqslant w(F) - 1$ for every $F \in \mathscr{F}_k$, $F \neq A_k$. Thus, in this case A_k is a unique minimum weight set in $\mathscr{F}_k$. Conversely, if there is a positive weighting that satisfies the conditions of Task B, it must clearly satisfy (2). If (1) is possibly violated, then after multiplying the solution by $1/w_{\min}$ all the inequalities will be satisfied, where $w_{\min}$ is the minimum of the positive weights. Note that the size of the factor $1/w_{\min}$ does not matter, since it merely serves to prove that the solvability of Task B implies the solvability of the system (1) and (2).

This system of linear inequalities, however, can contain exponentially many inequalities and, additionally, they may not even be explicitly listed. Nevertheless, we can still apply those linear programming algorithms that do not need an explicit list of the inequalities, they can work with a *separation oracle*. A well known such method is the Ellipsoid Algorithm, the first polynomial-time solution for linear programming (for an overview see e.g. [19]). It is known that the Ellipsoid Algorithm with separation oracle runs in time that is polynomially bounded in terms of the number of variables, the number of bits that describe any given inequality, and the running time of the separation oracle. In our case this means that for any given vector w we should be able to check in polynomial time, using the algorithms $\mathscr{A}_k$ as subroutines, whether w satisfies (1) and (2), or, if it does not, we have to find a violated inequality. If we can do this, then the Ellipsoid Algorithm finds the solution in (oracle) polynomial time, since our inequalities have all $0, 1, -1$ coefficients and constants.

Now let w be a vector. We can directly check if it satisfies (1). If it does not, then we have directly found at least one violated inequality in (1), so suppose $w \geqslant 1$ holds. Let us take an index k. We check (2) depending on which of $k > i_0$, $k \leqslant i_0$ holds (and we do this for each k):

Case 1: $k > i_0$. In this case we run algorithm $\mathscr{A}_k$ with input w. Let A be the returned minimum weight set. We compare $w(A_k)$ and $w(A)$. If $w(A_k) \leqslant w(A)$ (in fact, only equality can hold), then A_k is a minimum weight set in $\mathscr{F}_k$, so all inequalities with this k must be satisfied in (2). On the other hand, if $w(A_k) > w(A)$, then this provides a violated inequality, namely

$$\sum_{j \in A} w_j - \sum_{j \in A_k} w_j \geqslant 0$$

is violated.

Case 2: $k \leqslant i_0$. Now we start again as in Case 1. If $w(A_k) > w(A)$, then we found a violated inequality, since with $s(i_0 - k) = 1$

$$\sum_{j \in A} w_j - \sum_{j \in A_k} w_j \geqslant 1 \tag{3}$$

is violated. If $w(A_k) = w(A)$, i.e., A_k has minimum weight in $\mathscr{F}_k$, but $A_k \neq A$, then we again found a violated inequality, since (3) is still not satisfied, the left-hand side being 0. The remaining possibility is $A_k = A$. In this case, we have to check whether for any $A' \in \mathscr{F}_k$ the inequality $w(A') \geqslant w(A_k) + 1$ holds whenever $A' \neq A_k$. This can be

done as follows. Let x be an element of A_k. For each such x we define a new weight vector $w^{(x)}$ defined by increasing the weight on x by 1 and leaving the rest unchanged, i.e., we introduce $|A_k|$ new weight vectors, one for each $x \in A_k$, by

$$w_i^{(x)} = \begin{cases} w_i + 1 & \text{if } i = x, \\ w_i & \text{if } i \neq x. \end{cases} \tag{4}$$

Let us now run algorithm $\mathscr{A}_k$ with each of the new weight vectors and denote by B^x the returned minimum weight set when $w^{(x)}$ is used. We claim that $w(A') \geqslant w(A_k) + 1$ holds for every $A' \in \mathscr{F}_k$, $A' \neq A_k$, if and only if

$$\forall x \in A_k\colon\ w^{(x)}(B^x) \geqslant w(A_k) + 1. \tag{5}$$

To show this let A_0 be a set with minimum $w(A_0)$ in $\mathscr{F}_k - \{A_k\}$, i.e., it is a "second minimum" in $\mathscr{F}_k$. Assume $w(A_0) < w(A_k) + 1$. Then, by $w \geqslant 1$, $A_k \subseteq A_0$ cannot hold, so there must be an $x_0 \in A_k$ with $x_0 \notin A_0$. For this x_0 we have $w^{(x_0)}(A_k) = w(A_k) + 1 > w(A_0) = w^{(x_0)}(A_0) \geqslant w^{(x_0)}(B^{x_0})$, implying $B^{x_0} \neq A_k$ and $w^{(x_0)}(B^{x_0}) = w(B^{x_0})$, by the special choice of A_0. Hence, (5) is not satisfied and $w(B^{(x_0)}) < w(A_k) + 1$ generates a violated inequality. Conversely, if (5) is satisfied, then no such A_0 with $w(A_0) < w(A_k) + 1$ can exist, so all inequalities in (2) are satisfied for this index k.

Thus, doing the above for all k, we can check whether w satisfies (1) and (2) and, if not, we find a violated inequality. The procedure requires running the algorithms $\mathscr{A}_k$ at most nm times, along with simple recordkeeping, so we constructed a separation oracle that works in polynomial time if the $\mathscr{A}_k$ are used as oracles. The rest of the work is done by the Ellipsoid Algorithm. □

4. Finding a weighting with minimum error when no exact solution exists

As we have seen in the previous section, all of our considered inverse problems, along with many conceivable similar ones, can be cast in a general framework. An important question still remains, however: what can be done if no weight vector exists that satisfies the requirements? From the practical point of view it is desirable that if no w can satisfy the conditions exactly, then we at least aim at providing one that satisfies them with minimum error, rather than not finding anything at all. In the present section we show how to find such a solution in a general framework, such that the solution is optimal in the sense that it minimizes the maximum absolute error.

Let us define the problem version we consider for this situation, using the notations of the previous section.

Task C

Input: Subsets $A_1 \in \mathscr{F}_1, \ldots, A_m \in \mathscr{F}_m$.

Find: A weight vector $w \geqslant 1$, that minimizes the maximum absolute error

$$E = \max_i |w(A_i) - \mathscr{A}_i(w)|.$$

Note that the constraint $w \geqslant 1$ is needed for normalization purposes, otherwise one could trivially make the error arbitrarily small by assigning sufficiently small positive weights to everything. We also note that the absolute value can be omitted from the definition of E, as $w(A_i) \geqslant \mathscr{A}_i(w)$ always holds, being $A_i \in \mathscr{F}_i$.

Theorem 2. *There exists an algorithm $\mathscr{A}$ with the following properties*:

- *Algorithm $\mathscr{A}$, using the given algorithms $\mathscr{A}_1,\ldots,\mathscr{A}_m$ as subroutines, finds an optimal weighting w for Task C, i.e., w minimizes the maximum absolute error under the conditions of Task C.*
- *If every call to $\mathscr{A}_i$, $i=1,\ldots,m$, is counted as a single step, then algorithm $\mathscr{A}$ runs in polynomial time.*

Proof. Let us use the same notations as in Theorem 1 and consider the following linear program:

$$y \to \max!, \tag{6}$$

$$y \leqslant 0, \tag{7}$$

$$w_i \geqslant 1 \quad (i=1,\ldots,n), \tag{8}$$

$$\sum_{j\in F} w_j - \sum_{j\in A_k} w_j \geqslant y \quad (\forall F \in \mathscr{F}_k - \{A_k\},\ k=1,\ldots,m). \tag{9}$$

Note that (8) and (9) are the same as (1) and (2), with the only change that $s(i_0-k)$ is replaced by the new variable y. On the other hand, these constraints can always be satisfied, since for any non-negative weighting with $y=-w(H)$ (9) becomes $w(F)\geqslant w(A_k)-w(H)$ which is always true.

Now, similarly to Theorem 1, we can again construct a separation oracle, as follows. Given a vector (w,y), first we directly check if (7) and (8) are satisfied. If not, we have directly found a violated inequality. If they are satisfied, then we proceed as follows. For each index k run algorithm $\mathscr{A}_k$ with input w. Assume that for a given k the returned minimum weight set is A. Let us compare $w(A_k)$ and $w(A)$. If $w(A)\geqslant y+w(A_k)$ then all inequalities with this k must be satisfied in (9), since A is a minimum weight set. On the other hand, if $w(A)<y+w(A_k)$, then this provides a violated inequality, namely

$$\sum_{j\in A} w_j - \sum_{j\in A_k} w_j \geqslant y$$

is violated.

Having constructed the separation oracle, we can find an optimum solution (w_0,y_0) to the linear program by the Ellipsoid Algorithm. It follows from the construction that $|y_0|$ is the smallest possible value for which $w_0(A_k)$ differs form $\mathscr{A}_k(w_0)$ $(\forall k)$ at most by $|y_0|$, which proves the theorem. □

5. A case study using unique inverse shortest paths

To support applicability in telecommunications, in this section we present a case study that uses the unique inverse shortest path problem (Task II in the Introduction). First in Section 5.1 we show that using the special structure the unique inverse shortest path problem can be solved by a simpler and more direct LP formulation than the one which would follow from the general framework. Then, in Section 5.2 we apply

the solution to demonstrate that one can achieve improved network performance in a realistic scenario. This is shown via a simulated hypothetical pan-European ATM network with randomly generated traffic.

5.1. A direct LP solution for the unique inverse shortest path problem

Using the special structure of shortest paths we show that it allows a direct LP formulation that is simpler than the separation oracle-based formulation in the general framework. To describe the details let us define two sets of variables. For each we also provide the intended meaning that will help in understanding the constraints (note that these are *variables*, that is, they do not have a known value at the beginning):

w_{ij}: the weight of the edge (i,j). If the pair $e=(i,j)$ is denoted by a single letter e, then the notation w_e will be used.

d_{ij}: the distance (= weight of a min-weight path) between i and j.

Naturally, if $i=j$ then $d_{ii}=0$ is expected from the intended meaning, therefore, these values are set in advance.

For the easy description of some of the inequalities we introduce further notations. Let us say that an edge $e=(k,l)$ is a *shortcut* for a path $P_{ij}\in\mathscr{P}$ if e connects two non-consecutive nodes k,l on the path. The set of shortcut edges for the path is denoted by $S(P_{ij})$ and so the relationship that e is a shortcut for P_{ij} is denoted by $e\in S(P_{ij})$. Further, let $B(e,P_{ij})$ denote the subset of edges on P_{ij} that are bridged by e, i.e., those edges of P_{ij} that are between the two endpoints of e.

Now let us define the following sets of linear inequalities ("s.t." stands for "such that"):

$$d_{ij}\geqslant 0 \quad (\forall i,j), \tag{10}$$

$$w_{ij}\geqslant 1 \quad (\forall e=(i,j)\in E), \tag{11}$$

$$d_{ik}+w_{kj}\geqslant d_{ij} \quad (\forall i,j,k;\ \text{s.t. } (k,j)\in E), \tag{12}$$

$$\sum_{e\in P_{ij}} w_e \leqslant d_{ij} \quad (\forall i,j;\ \text{s.t. } P_{ij}\in\mathscr{P}), \tag{13}$$

$$d_{ik}+d_{kj}\geqslant d_{ij}+1 \quad (\forall i,j,k;\ \text{s.t. } P_{ij}\in\mathscr{P},\ k\notin P_{ij}), \tag{14}$$

$$w_e\geqslant 1+\sum_{f\in B(e,P_{ij})} w_f \quad (\forall e,i,j;\ \text{s.t. } P_{ij}\in\mathscr{P},\ e\in S(P_{ij})). \tag{15}$$

Using these inequalities the unique inverse shortest path problem can be characterized as follows.

Theorem 3. *Task* II *has a solution if and only if system* (10)–(15) *of linear inequalities has a solution. Moreover, in every solution of the linear inequality*

system, the values of the w_e variables define a weighting that satisfies the requirements of Task II.

Proof. Assume first that systems (10)–(15) has a solution. Let us denote a solution by w_e^*, d_{ij}^*. Further, let D_{ij}^* be the distance between nodes i and j, that is, the weight of a minimum weight path between i and j with respect to the weights w_e^*.

First we show that $d_{ij}^* \leqslant D_{ij}^*$ holds. Assume that indirectly this is not true and take a pair (i,j) for which $d_{ij}^* > D_{ij}^*$ and D_{ij}^* is the smallest possible among the counterexamples. Since $d_{ii}^* = D_{ii}^* = 0$, therefore, $i \neq j$ must hold. Now take a minimum weight path P from i to j, that is, the weight $w^*(P)$ of the path is equal to D_{ij}^*. Let k be the node on the path directly preceding j ($k = i$ may possibly hold). Since on the path k precedes j, therefore with the positive weights $D_{ik}^* < D_{ij}^*$ holds. This implies $d_{ik}^* \leqslant D_{ik}^*$ because D_{ij}^* was the smallest counterexample with $d_{ij}^* > D_{ij}^*$, so for the smaller D^* values the inequality must be already in the desired "$\leqslant$" direction. Now, using the fact $D_{ik}^* + w_{kj}^* = D_{ij}^*$ and the assumption $d_{ij}^* > D_{ij}^*$, combined with (12), we obtain

$$d_{ij}^* \leqslant d_{ik}^* + w_{kj}^* \leqslant D_{ik}^* + w_{kj}^* = D_{ij}^* < d_{ij}^*,$$

a contradiction. This proves that $d_{ij}^* \leqslant D_{ij}^*$ holds for all i,j. Now, for any path P_{ij}, the combination of (13) with $d_{ij}^* \leqslant D_{ij}^*$ yields

$$d_{ij}^* \geqslant \sum_{e \in P_{ij}} w_e^* = w^*(P_{ij}) \geqslant D_{ij}^* \geqslant d_{ij}^*,$$

which can only hold with equality throughout, implying $w^*(P_{ij}) = D_{ij}^*$. Thus, P_{ij} must be a minimum weight path with respect to the weighting w_e^*.

To show the uniqueness suppose indirectly that for some path $P_{ij} \in \mathscr{P}$ there is another path P'_{ij} between the same endpoints with equal weight. Now we have to consider two possible cases.

Case 1: Assume P'_{ij} has a node k that is not on P_{ij}. The node k divides P'_{ij} into two subpaths, let these be A and B, in the order of traversal from i. Then we have

$$w^*(P'_{ij}) = w(A) + w(B) \geqslant D_{ik}^* + D_{kj}^* \geqslant d_{ik}^* + d_{kj}^* \geqslant d_{ij}^* + 1 = w^*(P_{ij}) + 1,$$

contradicting to $w^*(P'_{ij}) = w^*(P_{ij})$.

Case 2: Each node of P'_{ij} is also on P_{ij}. Then, moving along on P'_{ij} from i to j, in each step we either move along an edge of P_{ij} or we bridge over certain edges of P_{ij} (forward or backward). Since at least one of the moves must be such a shortcut (otherwise the two paths would coincide) and by (15) the shortcut has strictly larger weight than the bridged subpath; therefore, we have $w^*(P'_{ij}) > w^*(P_{ij})$ (taking into account that each edge of P_{ij} must be either bridged over at least once or traversed directly). This contradicts again to the indirect assumption.

The above reasoning shows that we must have unique shortest paths. Conversely, if there is a weighting that provides a solution to Task II, then one can directly check that all the inequalities have to be satisfied, except possibly (11). Since, however, multiplying the (positive) weights by any positive constant does not change the properties, therefore, (11) can also be satisfied by multiplying the solution with $1/w_{\min}$, where $w_{\min}$ is the minimum weight that occurs in the weighting. In this way a solution is obtained to the full system of linear inequalities, which proves the theorem. □

Remarks.

- The size of the $1/w_{\min}$ factor does not matter, since it is needed only to prove that the solvability of Task II implies the solvability of the system of linear inequalities, but it is not needed when we actually solve the system.
- The known solving algorithms for linear programming are not *strongly polynomial*, i.e., their running time depends not only on the dimension of the problem, but also on the size of numbers in the input, even if the elementary arithmetic and logical operations are done in unit time, independently of the sizes of the numbers. This can happen, however, only if the input can contain numbers that are exponentially large in terms of the number of variables and constraints. One can observe that all the coefficients and constants in (10)–(15) are 0, 1 or -1, which implies that this system can be solved in strongly polynomial time. In fact, a weaker condition would be already enough: it is known [22] that if the coefficients in the matrix are small but the constants are arbitrary, the system can still be solved in strongly polynomial time (see also [19, Chapter 15.2]).
- From formulation (10)–(15) one can directly see that for a graph with n vertices and m edges the number of variables is $m + n(n-1)/2$ and the number of linear inequalities is $O(\min(mn^2, n^4))$.

5.2. A simulated network example

To gain some experience on how a network behaves if routing is done using our methods and to judge whether one can achieve improved network performance in a realistic scenario, we simulated a 15-node hypothetical pan-European ATM network with randomly generated traffic demands, using the unique inverse shortest path formulation presented in the preceding subsection. The simulation was done with *PLASMA*, a PNNI-capable ATM network simulator of Ericsson [11].

The graph that describes the network topology is shown in Fig. 1. We compared two situations. In the first version, a shortest (minimum-hop) path was fixed between each origin–destination pair and the network was dimensioned with the assumption that the traffic will travel on these fixed routes. Having dimensioned the network this way, in the simulation the actual routes were chosen using unit Administrative Weights in PNNI. The unit weights, naturally, resulted in minimum-hop paths, but not necessarily the same ones that were assumed in the dimensioning.

In the second version we used the same dimensioning phase, i.e., everything was the same as in the first version, except that in the simulation the Administrative Weights were computed according to Task II. That is, due to the weighting that generated *unique* shortest paths, PNNI was forced to always choose the preset fixed routes, as opposed to the first version when any of the parallel minimum-hop paths could be chosen.

We measured in both cases the average blocking probability in the network and the average load on each link. The average blocking probability in the first case was 20%, while in the second case it went down to 3.5%.

The average loads on links are shown in Fig. 2. It is interesting to observe that the optimized weights (second case) provide much better load balancing. On the other

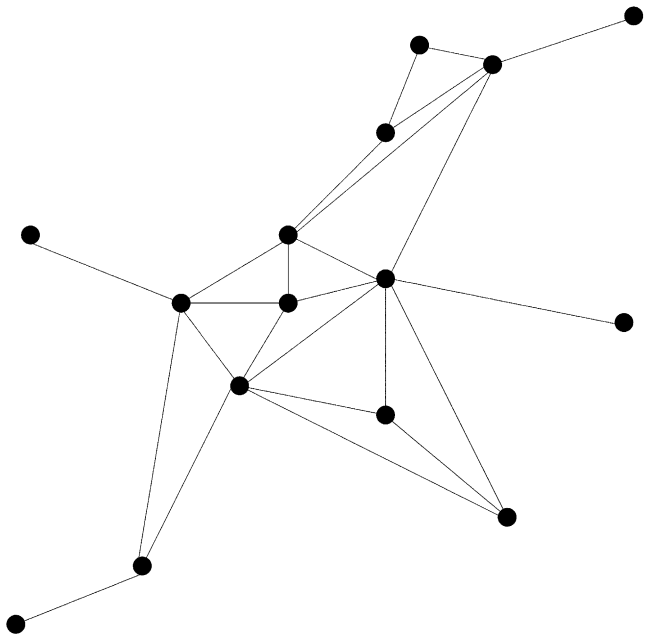

Fig. 1. Network topology for simulation.

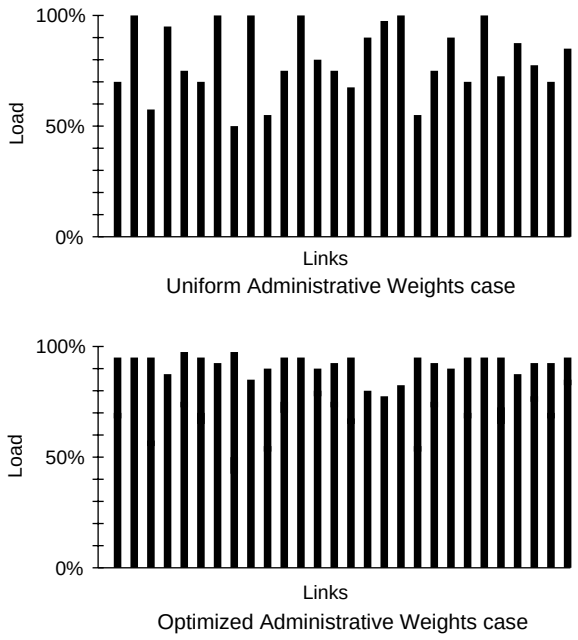

Fig. 2. Average link loads.

hand, for certain links, this can result in increasing average load which implies higher blocking probability. Nevertheless, the *overall* network performance was dramatically improved, as shown by pushing down the average blocking probability from 20% to 3.5%.

Note that the only difference between the two cases was that in the second one traffic could not deviate from the fixed routes because the Administrative Weights were chosen to generate the preset routes in a unique manner. This example shows that it can bring real practical gain in network performance (and, therefore, in network revenue) if the inverse shortest path methodology is applied to guide the routing protocol.

6. Conclusion

We have presented solutions to various versions of general inverse combinatorial optimization tasks, motivated by inverse shortest path problem versions. We have shown that all the tasks can be handled by linear programming, even in the generalized case when little is known about the inherent structure. We also pointed out the relevance of this problem family to high-speed telecommunication networks and demonstrated through simulation the direct gain that can be obtained using our approach in PNNI-based ATM networks.

Acknowledgements

The authors are grateful for the continuous support and encouragement of Dr. Miklós Boda, Head of Traffic Lab, Ericsson, Dr. Géza Gordos, Head of the Department of Telecommunications and Telematics, Technical University of Budapest, and Dr. Tamás Henk, Head of the High Speed Networks Lab at the same Department.

References

[1] R.K. Ahuja, J.B. Orlin, Solving the convex ordered set problem with applications to isotonic regression, Working Paper SWP# 3988, MIT Sloan School of Management, Cambridge, MA, 1997.
[2] R.K. Ahuja, J.B. Orlin, Inverse optimization, Part I: linear programming and general problem, Working Paper SWP# 4002, MIT Sloan School of Management, Cambridge, MA, 1998.
[3] R.K. Ahuja, J.B. Orlin, A fast algorithm for the bipartite node weighted matching problem on path graphs with application to the inverse spanning tree problem, Working Paper SWP# 4006, MIT Sloan School of Management, Cambridge, MA, 1998.
[4] D. Burton, B. Pulleyblank, Ph.L. Toint, The inverse shortest path problem with upper bounds on shortest paths costs, in: P. Bartalos, D.W. Hear, W.H. Hager (Eds.), Network Optimization, Lecture Notes in Economics and Mathematical Systems, Vol. 450, pp. 156–171.
[5] D. Burton, Ph.L. Toint, On an instance of the inverse shortest path problem, Math. Programming 53 (1992) 45–61.
[6] D. Burton, Ph.L. Toint, On the use of an inverse shortest path algorithm for recovering linearly correlated costs, Math. Programming 63 (1994) 1–22.
[7] M. Cai, X. Yang, Inverse shortest path problems, Technical Report, Institute of Systems Sciences, Academia Sinica, Beijing, China, 1994.

[8] M. Cai, X. Yang, Y. Li, Inverse polymatroidal flow problem, Research Report, Institute of Systems Science, Chinese Academy of Science, Beijing, China, 1996.
[9] A. Faragó, S. Blaabjerg, L. Ast, G. Gordos, T. Henk, A new degree of freedom in ATM network dimensioning: optimizing the logical configuration, IEEE J. Selected Areas Commun. 13 (1995/7) 1199–1206.
[10] A. Girard, Routing and Dimensioning in Circuit-switched Networks, Addison-Wesley, Reading, MA, 1990.
[11] Zs. Haraszti, I. Dahlquist, A Faragó, T. Henk, PLASMA—an integrated tool for ATM network operation, International Switching Symposium (ISS'95), Berlin, 1995, pp. 314–318.
[12] Z. Hu, Z. Liu, A strongly polynomial algorithm for the inverse shortest arborescence problem, Working Paper, Institute of Systems Sciences, Academia Sinica, Beijing, China, 1995.
[13] S. Hunag, Z. Liu, On the inverse problem of k-matching of bipartite graphs, Working Paper, School of Business and Management, Hong Kong University of Science and Technology, Hong Kong, 1995.
[14] S. Hunag, Z. Liu, On the inverse version of the minimum cost flow problem, Working Paper, School of Business and Management, Hong Kong University of Science and Technology, Hong Kong, 1995.
[15] Y. Li, Inverse combinatorial optimization problems and connected factor problems, Dissertation, Institute of Systems Science, Chinese Academy of Science, Beijing, China, 1996.
[16] Z. Ma, S. Xu, J. Zhang, Algorithms for inverse minimum spanning tree problem, Working Paper, Department of Mathematics, City Polytechnic of Hong Kong, Hong Kong, 1996.
[17] Private Network–Network Interface Specification, Version 1.0 (PNNI 1.0), The ATM Forum, Technical Committee, March 1996.
[18] J. Roberts, U. Mocci, J. Virtamo (Eds.), Broadband Network Traffic, Springer, Berlin, 1996.
[19] A. Schrijver, Theory of Linear and Integer Programming, Wiley, New York, 1990.
[20] P.T. Sokkalingam, The minimum cost flow problem: primal algorithms and cost perturbations, Dissertation, Department of Mathematics, Indian Institute of Technology, Kanpur, India, 1996.
[21] P.T. Sokkalingam, R.K. Ahuja, J.B. Orlin, Solving the inverse spanning tree problems through network flow techniques, Working Paper SWP# 3890, MIT Sloan School of Management, Cambridge, MA, 1996.
[22] É. Tardos, A strongly polynomial algorithm to solve combinatorial linear programs, Oper. Res. 34 (1986) 250–256.
[23] S. Xu, J. Zhang, An inverse problem of the weighted shortest path problem, Japan J. Ind. Appl. Math. 12 (1995) 47–59.
[24] C. Yang, J. Zhang, Z. Ma, Inverse maximum flow and minimum cut problem, Optimization 40 (1997) 147–170.
[25] J. Zhang, M.C. Cai, Inverse problem of minimum cuts, Math. Methods Oper. Res. 47 (1998) 1.
[26] J. Zhang, Z. Ma, C. Yang, A column generation method for inverse shortest path problems, ZOR-Math. Methods Oper. Res. 41 (1995) 347–358.